State Lock Auctions: Towards Collaborative Block Building

Many thanks to Conor, Quintus, Louis, and Shea for discussions and comments that inspired this post.

Cross posted at State Lock Auctions: Towards Collaborative Block Building - Research - The Flashbots Collective

[

BJ506v_E2.jpg

1024×1024 266 KB

](https://ethresear.ch/uploads/default/original/2X/2/2da9ab0480ae7a3884364b6090b66a9cbcaadad3.jpeg)

Overview

Due to the public first-price nature of theMEV-boost auction, searchers are incentivized to vertically integrate, becoming "searcher-builders" to bid "more strategically". This verticalization aimed at gaining an edge in the MEV-Boost auction has had the unfortunate consequence of considerably raising the barriers to entry for statistical arbitrage, and as a follow on consequence, has created an unfavorable market structure where new "searcher-builders" are bulk purchasing orderflow from other builders and subsidizing blocks as an effective "advertisement spend" to reach orderflow parity with incumbent block builders. Attempts to remove this edge from "strategic bidding" have primarily been focused on modifying the auction into a sealed-bid format, all of which have fallen short as they introduce new collusion vectors in the absence of private and credible commitment technology such as MPC, FHE, and TEEs.

Greatly borrowing from distributed systems literature, this post sketches a proposal for enabling the ability to "lock state" through an open auction in the mev-boost protocol, with a follow-on feature of providing visibility into this "locked state". The ultimate outcome of this approach is that block builders are no longer effectively "pipeline stalled" via uncertainty in constructing the rest of the block, which opens the door for "Collaborative Block Building". In distributed systems terms, we are creating a "boot leg" decentralized multi-version concurrency control system for the ethereum distributed state machine by leveraging visibility.

The main advantages of this approach are that it maintains basic privacy and bid update-ability to searchers competing in these high-velocity arbitrages, supports more generic arbitrage compared to previous "laned" approaches, and is easily deployable into the current market. The main drawbacks are its creation of a "pseudo-enshrinement" of single transaction arbitrages at the top of the block and offering a new censorship and griefing vector. As this proposal is merely a sketch, there are still large unknowns around how to effectively price the cost to lock state, the effects on searcher's bidding strategies, and how to generalize the auction past the ability to lock a single set of states.

Desirable Properties for a State Lock Auction

While not exhaustive, some immediately apparent favorable properties:

1. Locker Safety
2. Revealing the address of the person who locked the state before the block is confirmed creates a targeted DoS vector.
3. Bid Update-ability
4. A searcher who commits to a state lock should be able to update the parameters of their transaction as long as it does not modify the state access list.
5. Locked State Visibility
6. The storage slots of a locked state should be publicly visible to block builders.
7. Guaranteed Lock Payment
8. Whether a searcher updates the bid or not, a payment transaction should be included in the block for locking state.
9. Squatting Resistance
10. Ensuring the cost to lock up extensive amounts of state is prohibitively high to prevent griefing.
11. Contention Aware
12. The price of a lack should reflect the amount of interest in writing to that state.

Access Lists

For most of this post we refer to the state to be locked as an “access list”. In the context of EIP-2930, a state access list A

is defined as a sequence of tuples:

A = [(\text{address}, [\text{storageKey}_1, \text{storageKey}_2, \ldots]), \ldots]

Where:

- \text{address}

is an Ethereum address of a contract or an Externally Owned Account (EOA) that the transaction interacts with.

- [\text{storageKey}_1, \text{storageKey}_2, \ldots]

represents a list of storage keys within the specified contract’s storage the transaction will access.

An example access list for a swap is shown in Appendix A.

Proposal: Relay Run Single Lock Auction

Representing searchers as S

, the relay as R

, and partial block builders as PB

, the auction process unfolds as follows:

1. Open Auction

:

- Prior to the cutoff time T_{cutoff}

seconds into the slot, searchers submit their state lock commitments to R

with two distinct transactions:

\text{SubmitStateCommitment}(n, A, tx_{state}, tx_{payment}, b)

where

- n

: Slot number.

- A

: Access list.

- tx_{state}

: State access transaction. (read: “the arb”)

- tx_{payment}

: Payment transaction ensuring the lock cost is covered.

- b

: Bid value,

and upon submission R

validates:

- b

must meet or exceed the lock reserve cost based on A

. * The lock reserve cost is determined by:

- The lock reserve cost is determined by:

$$\text{LockCost}(A) = C \times |A|$$

where:

- C

represents a fixed cost per storage slot, and $|A|$

is the count of storage slots in the access list A

for tx_{state}

. This approach naively makes it more expensive to lock large amounts of state but is far from an ideal pricing rule.

-
    - tx_{state}

should be a transaction with access list equal to A

.

- tx_{state}

should be a transaction with access list equal to A

.

-
    - $tx_{payment}$

should be a transaction paying the validator from an EOA with no calldata.

- $tx_{payment}$

should be a transaction paying the validator from an EOA with no calldata.

-
    - (tx_{state}

, $tx_{payment}$

) simulated together at top-of-block pay the amount promised by b

.

- (tx_{state}

, $tx_{payment}$

) simulated together at top-of-block pay the amount promised by b

.

- n

: Slot number.

- A

: Access list.

- tx_{state}

: State access transaction. (read: “the arb”)

- $tx_{payment}$

: Payment transaction ensuring the lock cost is covered.

- b

: Bid value,

- b

must meet or exceed the lock reserve cost based on A

. * The lock reserve cost is determined by:

- The lock reserve cost is determined by:
- C

represents a fixed cost per storage slot, and |A|

is the count of storage slots in the access list A

for tx_{state}

. This approach naively makes it more expensive to lock large amounts of state but is far from an ideal pricing rule.

-
    - tx_{state}

should be a transaction with access list equal to A

.

- tx_{state}

should be a transaction with access list equal to A

.

-
    - tx_{payment}

should be a transaction paying the validator from an EOA with no calldata.

- tx_{payment}

should be a transaction paying the validator from an EOA with no calldata.

-
    - (tx_{state}

, tx_{payment}

) simulated together at top-of-block pay the amount promised by b

.

- (tx_{state}

, tx_{payment}

) simulated together at top-of-block pay the amount promised by b

.

1. Winner Selection

:

- At T_{cutoff}

, R

picks the highest bid:b_{max} = \max({b_i | b_i \geq \text{LockCost}(A_i), \forall i \in S})

- R

then discloses the winning access list A_{max}

associated with b_{max}

to anyone who calls:

- R

then discloses the winning access list A_{max}

associated with b_{max}

to anyone who calls:

\text{GetLockedState}(n)

1. Bid Updating and Partial Block Submission

:

Two main activites can occur post winner selection:

- The winning searcher addr_{max}

can update their state access transaction facilitated through:

\text{UpdateStateCommitment}(n, addr_{max}, A_{max}, tx_{state_new}, b_{new})

where:

- n

must equal to the most recent slot associated with the winning bid from addr_{max}

- R

must validate that the updated state access transaction tx_{state_new}

has an acess list eqal to A_{max}

.

- and the payment transaction tx_{payment}

implicitly remains unchanged.

- n

must equal to the most recent slot associated with the winning bid from addr_{max}

- R

must validate that the updated state access transaction tx_{state_new}

has an acess list eqal to A_{max}

.

- and the payment transaction tx_{payment}

implicitly remains unchanged.

- Partial block builders query GetLockedState(n)

for A_{max}

, build partial blocks based on this information, and submit them using:

\text{SubmitPartialPayload}(TX_{list}, b_{PB})

where:

- TX_{list}

includes a list of transactions

- b_{PB}

represents the bid for the partial block.

and upon receiving a partial payload submission, the relay (R

) undertakes the following steps:

- 1. Transaction Combination

: R

creates

\text{CollabBlock = }(tx_{state}, tx_{payment},TX_{list})

- 1. Block Validation

: R

simulates CollabBlock

to perform usual block validation as well as ensures the combined block bid value is equal to b_{max} + b_{PB}

.

- 1. Payment Handling

: R

crafts, signs, and inserts payment transaction to pay validator and partial block builder

- 1. State Root Calculation

: If the validation is successful, R

calculates the block’s state root and related fields to ensure compatibility with the latest EL and CL specs, inserts into a beacon block with latest payload attributes, and makes available to getHeader

calls from validators.

- TX_{list}

includes a list of transactions

- b_{PB}

represents the bid for the partial block.

- 1. Transaction Combination

: R

creates

- 1. Block Validation

: R

simulates CollabBlock

to perform usual block validation as well as ensures the combined block bid value is equal to b_{max} + b_{PB}

.

- 1. Payment Handling

: R

crafts, signs, and inserts payment transaction to pay validator and partial block builder

- 1. State Root Calculation

: If the validation is successful, R

calculates the block's state root and related fields to ensure compatibility with the latest EL and CL specs, inserts into a beacon block with latest payload attributes, and makes available to getHeader

calls from validators.

Below is an artist's rendition of the above auction process.

[

Screenshot 2024-02-03 at 3.58.51 PM

1850×1312 253 KB

](https://ethresear.ch/uploads/default/original/2X/9/9a909c841f7f1cd6d98a63a5ce4364907ee8d223.jpeg)

A rough sketch of the changes to the relay spec can be seen inAppendix B.

Analysis

This approach meets most of our core objectives for a state lock system: ensuring Locker Safety

, Bid Update-ability

, Locked State Visibility

, and Guaranteed Lock Payment

, takes an initial stab at Squatting Resistance

, and makes no attempt at being Contention Aware

. Even further, Locked State Visibility

lays the foundation for collaborative block building by allowing builders to append partial blocks to top-of-block arbitrages more confidently. The design deliberately avoids checks for conflicting transactions during SubmitPartialPayload

to avoid blocking any other transactions also intending to touch the contentious locked state. Block builders are free to send in blocks with conflicts, but they are now aware of conflicts beforehand, which is an improvement over today's market and creates an incentive to participate. Searchers are also incentivized to participate as it makes entry to stat arb easier and increases the likelihood of block inclusion. Relays take on extra work and, therefore, additional cost, but this approach normalizes the ability to take profit through a second price auction between relays in a roundabout incentive.

One area of concern is shifting towards a more relay-centric block building andnormalizing builderelays. My biased and cautious initial reaction is that we should be okay with this trend as long as it does not create opinions on which blocks are built or give an unequal advantage to one party over another. From this POV, relays will continue to house more auction-related logic as a trusted third party amongst a network of builders, each additional piece of logic ideally helping to promote more collaborative block building.

This approach requires no protocol changes and can be run in parallel with the current mev-boost auction. Additionally, it creates a "laned" approach without restricting it to a single arbitrage type, but is not fully generalized as it restricts top-of-block arbitrages to a single transaction. Exploring bundle support is thus an ideal future exploration, but if done naively, could create additional complications via a new avenue to submit entire blocks.

There are, however, a few complexities with taking this approach live. One is that it suffers from a cold start problem. If, at launch, only one searcher has developed the capabilities to compete in this auction, they will likely dominate for a short period and very cheaply, depending on the lock reserve cost price. The dual of this is that it only takes one searcher's participation to snowball the rest. Another area for improvement is that this design only focused on the singular relay case, but in reality, there are many relays that all act independently. One thought is that maybe synchronization doesn't matter since bid cancellations are a similar cross-relay problem that is not guaranteed In the multi-relay scenario, the best case is that each agrees on the state to be locked. In the worst case, each has a different state set and partial block builders construct for all potential paths. This creates more work, but at the very least, it is parallelizable and still a significant improvement over the status quo. Another area for future exploration is how this approach plays with optimistic relaying given its more complicated validation logic.

Lastly, the most significant unknowns here are the lock pricing mechanism and the state commitment's effects on searcher strategies. The lock pricing mechanism is very undefined and most likely the place where things can go comically wrong; therefore, follow-up work should try to reason about this analytically and adversarially. Incorporation of historical state

access into the pricing mechanism is also difficult because that is an extremely large vector to update if done naively and cannot be plugged into the block like the base fee relies on. Progress on figuring out this mechanism will also help to assess how big of a censorship vector this auction is and tease out the incentives more concretely. The effects of creating a new upfront cost to doing statistical arbitrage also should be thought through, but in the worst case, a searcher can effectively cancel their trade and simply pay the fee, not requiring them to commit to trading large amounts at a price they no longer find attractive based on signals on other domains.

Add Ons

- A randomized closing time could be used to discourage last-second bidding. But in the multiple indepedent relay case, this comes at the cost of creating different winners per relay.
- The state lock auction could employ a second-price rule to determine payment, encouraging truthful bidding. This could be achieved by inserting a refund transaction into the block for the difference, at the cost of 21k gas.
- EIP-2930 transaction types could be used for the state access transaction, but they do support access to non-acces-list supplied storage slots so would not enable blind trust. In general moving in this direction will help streamline all future collaborative building efforts.
- The block construction in the relay provides an ideal time to plug inunconditional inclusion lists.
- Leave bidding open to replace the state access transaction with your own as long as it touches the same state as the original. The incenvites around this are a bit unclear.
- Allow an auction per overlapping access list in parallel.
- One challenge here is that an access list which represents arbitraging two pools can be broken into two auctions to arbitrage each pool, and the cost of locking each pool individually may not be as high as someone trying to lock them both at the same time. The generalization thus requires some form of combinatorial auction that allows you to bid conditionally on winning another auction.
- Tarun has cast'ed about how the Jito Solana MEV Auction works in a similar manner here
- Tarun has cast'ed about how the Jito Solana MEV Auction works in a similar manner here
- Another challenge is that in the single auction case, the relay can be sure that a winning transaction will produce a certain access list. In the double auction case, we can no longer be certain of the 2nd transaction's access list solely by looking at the first transaction's access list. This is exemplified by a 2nd transaction, which is conditional on a balance modified in transaction 1. Although we should be able to detect the possibility

of such a read-write conflict, it is most likely not stable enough ground to continuously stack these auctions on top of. A relay could simulate the 2nd transaction on top of the winner from the first auction, but this does not give us enough time in the block to run many auctions. Most likely, some constraints on the transaction's ability to write and read state will need to be employed to fix this.

- One challenge here is that an access list which represents arbitraging two pools can be broken into two auctions to arbitrage each pool, and the cost of locking each pool individually may not be as high as someone trying to lock them both at the same time. The generalization thus requires some form of combinatorial auction that allows you to bid conditionally on winning another auction.
- Tarun has cast'ed about how the Jito Solana MEV Auction works in a similar manner here
- Tarun has cast'ed about how the Jito Solana MEV Auction works in a similar manner here
- Another challenge is that in the single auction case, the relay can be sure that a winning transaction will produce a certain access list. In the double auction case, we can no longer be certain of the 2nd transaction's access list solely by looking at the first transaction's access list. This is exemplified by a 2nd transaction, which is conditional on a balance modified in transaction 1. Although we should be able to detect the possibility

of such a read-write conflict, it is most likely not stable enough ground to continuously stack these auctions on top of. A relay could simulate the 2nd transaction on top of the winner from the first auction, but this does not give us enough time in the block to run many auctions. Most likely, some constraints on the transaction's ability to write and read state will need to be employed to fix this.

Feasibility with ePBS

The challenge for integrating this approach with something like a future "ePBS" is that we would need to shift the auction to the attester layer, which naively requires an additional synchrony period. The network must come to an agreement on "what" the winning access list is, not to mention the p2p games that this incentivizes at the protocol layer.

One reason against moving this sort of thing into the protocol is that it doesn't affect the resources on the network from a

validator's perspective, more or less locks wont affect the resource requirements needed to perform a validator's duties. Additionally, lock auctions don't represent a liveliness risk to the network. If the auction goes down, then we degrade back to the current state of block building. An argument supporting moving it into the protocol is that the auction generates revenue from the protocol's state.

Conclusion

This proposal sketches out a design for a relatively crude mechanism to auction a top-of-block state lock on Ethereum and enables collaboration in the block builder market. While not an end-game solution, it creates an ideal interface for experimentation with state locks and unlocks

new aspects of the block building market, which may help us out of our current local maxima. Assuming an ideal lock pricing mechanism, this approach can go live into the wild quickly and move us one step towards a better builder market structure.

Appendix

A. Example Access List

Example Access List for

- tx hash 0x84ba67793c7aa0140ae9f7cbc58a7bec04099672d467dfe280ec5987b5f26b46 swapping Wrapped Ether

for Porktoshi Nakamoto

.

Created via Created via these python scripts.

[{ "address": "0xa65303cbe1186f58dda9cf96b29e1e89ae90b165", "storageKeys": [
"0x0000000000000000000000000000000000000000000000000000000000000008"] }, { "address":
"0xb39bc86ac75118011f646276fca48d56e54c4854", "storageKeys": [
"0x000000000000000000000000000000000000000000000000000000000000000d",
"0x7e3bb96fe9c3e8bf8baff39136a5e5778a1f21be90df3270983ce535ed884516",
"0x9db3014a7ecc5c8baca43505a03b4e7e24c2ec389a973d5605a299f04defc730"] }, { "address":
"0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2", "storageKeys": [
"0x377eec8667584e30e85471441b46e69393e5f35d2b63a0b4c26a5b1f6d059aa5"] }]

Here

- 0xC02aaA...3C756Cc2

is the WETH token contract

- 0xb39BC8...e54C4854

is the PORKTOSHI token contract

- 0xA65303...AE90B165

is the PORKTOSHI/WETH Uni V2 Pool

Based on the storage keys for the PORKTOSHI contract we can see there is some non-standard ERC20 behavior and the source code verifies that but is besides the point.

B. Rough Relay Spec Change

Validation rules and internal logic are explained in the previous sections.

1. relay_SubmitStateCommitment

Allows participants to submit a state commitment for a particular slot.

- Accepts:

- n

(Slot Number): Integer representing the slot number.

- A

(Access List): Array of objects detailing state keys (address

and storageKeys

) a transaction intends to interact with.

- statetx

(Signed Transaction): Hexadecimal string of the signed transaction.

- paymenttx

(Signed Transaction): Hexadecimal string of the signed transaction.

- b

(Bid Value): Integer representing the bid value in wei.

- n

(Slot Number): Integer representing the slot number.

- A

(Access List): Array of objects detailing state keys (address

and storageKeys

) a transaction intends to interact with.

- statetx

(Signed Transaction): Hexadecimal string of the signed transaction.

- paymenttx

(Signed Transaction): Hexadecimal string of the signed transaction.

- b

(Bid Value): Integer representing the bid value in wei.

- Returns:

Confirmation of submission.

2. relay_GetLockedState

Fetches the access list associated with the winning bid for the specified slot.

- Accepts:

- n

(Slot Number): Integer representing the slot number for which the access list is requested.

- n

(Slot Number): Integer representing the slot number for which the access list is requested.

- Returns:

- Access List (A

): Array of objects detailing state keys (address

and storageKeys

) for the top bid of the specified slot.

- Access List (A

): Array of objects detailing state keys (address

and storageKeys

) for the top bid of the specified slot.

3. relay_SubmitPartialPayload

Enables partial block builders to submit a set of transactions as a partial block proposal.

- Accepts:
- txList

(List of Transactions): Array of hexadecimal strings representing transactions.

- b

(Bid Value for Partial Block): Integer representing the bid value in wei for the partial block.

- txList

(List of Transactions): Array of hexadecimal strings representing transactions.

- b

(Bid Value for Partial Block): Integer representing the bid value in wei for the partial block.

- Returns:

Confirmation of partial payload submission.

[

manual mermaid diagram

1158×588 37.4 KB

](https://ethresear.ch/uploads/default/original/2X/8/8bb55ee74c55f92dd7de2d3269980497a6719e80.png)